合肥工業大學

《系统硬件综合设计》

实验报告

学生姓名 袁焕发

学　　号 2019217769

专业班级 物联网工程

指导教师 阙夏 丁贤庆 李建华

院系名称 计算机与信息学院

2022 年 1 月 04 日

目　录

目录

1. 实验目的

1. 掌握单周期 CPU 数据通路图的构成、原理及其设计方法；
2. 掌握单周期 CPU 的实现方法，代码实现方法；
3. 认识和掌握指令与 CPU 的关系；
4. 掌握测试单周期 CPU 的方法

2. 实验环境

Windows 11、Vivado 2020.1

3. 实验原理

3.1. 单周期 CPU

单周期 CPU 指的是一条指令的执行在一个时钟周期内完成，然后开始下一条指令的执行，即一条指令用一个时钟周期完成。电平从低到高变化的瞬间称为时钟上升沿，两个相邻时钟上升沿之间的时间间隔称为一个时钟周期。时钟周期一般也称振荡周期（如果晶振的输出没有经过分频就直接作为 CPU 的工作时钟，则时钟周期就等于振荡周期。若振荡周期经二分频后形成时钟脉冲信号作为 CPU 的工作时钟，这样，时钟周期就是振荡周期的两倍。

3.2. 执行过程

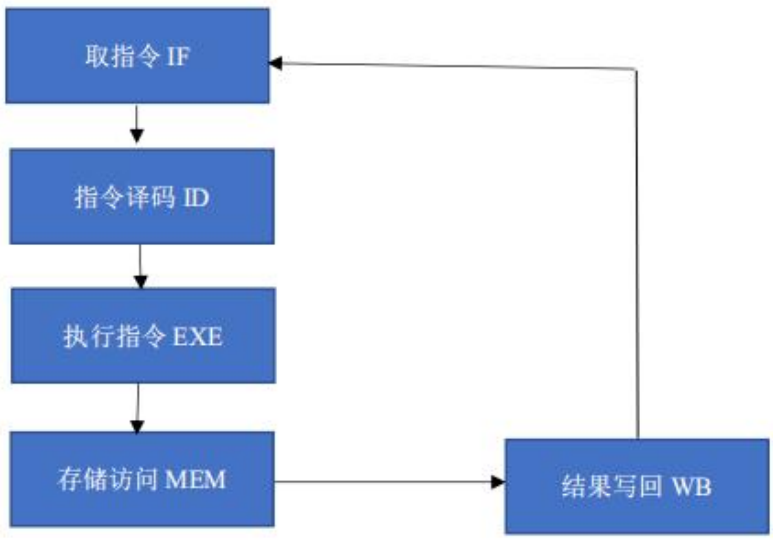


图 1 单周期 CPU 指令处理过程

(1) 取指令(IF)：根据程序计数器 PC 中的指令地址，从存储器中取出一条指令，同时，PC 根据指令字长度自动递增产生下一条指令所需要的指令地址，但遇到“地址转移”指令时，则控制器把“转移地址”送入 PC，当然得到的“地址”需要做些变换才送入 PC。

(2) 指令译码(ID)：对取指令操作中得到的指令进行分析并译码，确定这条指令需要完成的操作，从而产生相应的操作控制信号，用于驱动执行状态中的各种操作。

(3) 指令执行(EXE)：根据指令译码得到的操作控制信号，具体地执行指令动作，然后转移到结果写回状态。

(4) 存储器访问(MEM)：所有需要访问存储器的操作都将在这个步骤中执行，该步骤给出存储器的数据地址，把数据写入到存储器中数据地址所指定的存储单元或者从存储器中得到数据地址单元中的数据。

(5) 结果写回(WB)：指令执行的结果或者访问存储器中得到的数据写回相应的目的寄存器中。单周期 CPU，是在一个时钟周期内完成这五个阶段的处理。

3.3. 指令格式

MIPS 指令有以下三种形式：

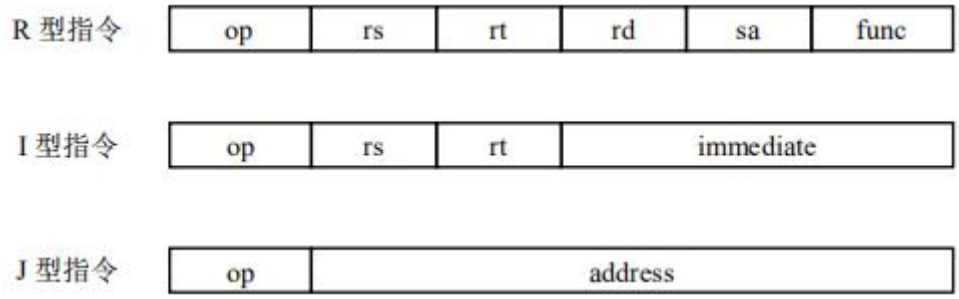


图 2 MIPS 指令

op：占六位，为操作码；

rs：只读。为第 1 个源操作数寄存器，寄存器地址是 00000~11111，00~1F；

rt：可读可写。为第 2 个源操作数寄存器，或目的操作数寄存器，寄存器地址 00000~11111；

rd：只写。为目的操作数寄存器，寄存器地址 00000~11111；

sa：为位移量（shift amt），移位指令用于指定移多少位；

funct：为功能码，在寄存器类型指令中（R 类型）用来指定指令的功能与

操作码配合使用；

immediate：为 16 位立即数，用作无符号的逻辑操作数、有符号的算术操作数、数据加载（Load）/数据保存（Store）指令的数据地址字节偏移量和分支指令中相对程序计数器（PC）的有符号偏移量；

address：为跳转地址。

控制信号名	状态“0”	状态“1”
Reset	初始化 PC 为 0	PC 接收新地址
PCWre	PC 不更改，相关指令：halt	PC 更改，相关指令：除指令 halt 外
ALUSrcA	来自寄存器堆 data1 输出，相关指令：add、sub、addiu、or、and、andi、ori、slti、beq、bne、bltz、sw、lw	来自移位数 sa，同时，进行(zero-extend)sa，即，相关指令：sll
ALUSrcB	来自寄存器堆 data2 输出，相关指令：add、sub、or、and、beq、bne、bltz	来自 sign 或 zero 扩展的立即数，相关指令：addiu、andi、ori、slti、sw、lw
DBDataSrc	来自 ALU 运算结果的输出，相关指令：add、addiu、sub、ori、or、and、andi、slti、sll	来自数据存储器（Data MEM）的输出，相关指令：lw
RegWre	无写寄存器组寄存器，相关指令：beq、bne、bltz、sw、halt	寄存器组写使能，相关指令：add、addiu、sub、ori、or、and、andi、slti、sll、lw
InsMemRW	写指令存储器	读指令存储器
mRD	输出高阻态	读数据存储器，相关指令：lw
mWR	无操作	写数据存储器，相关指令：sw
RegDst	写寄存器组寄存器的地址，来自 rt 字段，相关指令：addiu、andi、ori、slti、lw	写寄存器组寄存器的地址，来自 rd 字段，相关指令：add、sub、and、or、sll
ExtSel	(zero-extend)immediate（0	(sign-extend)immediate

	扩展），相关指令：andi、ori	（符号扩展），相关指令：addiu、slti、sw、lw、beq、bne、bltz

表 1 控制信号表

控制信号名	功能
PCSrc[1..0]	00：pc<－pc+4，相关指令：add、addi、sub、or、ori、and、slti、sll、sw、lw、beq(zero=0)、bne(zero=1)； 01：pc<－pc+4+(sign-extend) immediate，相关指令：beq(zero=1)、bne(zero=0)； 10：pc<－{(pc+4)[31:28], addr[27:2],2{0}}，相关指令：j； 11：未用
ALUOp[2..0]	000 Y = A + B 加，不带符号
	001 Y = A － B 减
	010 Y = B<<A B 左移 A 位
	011 Y = A ∨ B 或
	100 Y = A ∧ B 与
	101 Y=（A<B）?1: 0 比较 A 与 B
	110Y=(((rega<regb) && (rega[31] == regb[31])) ‖((rega[31] ==1 && regb[31]== 0))) ? 1:0，比较 A 与 B 带符号
	111 Y = A 异或 B

表 2 控制信号表补充

3.4. 模块以及说明

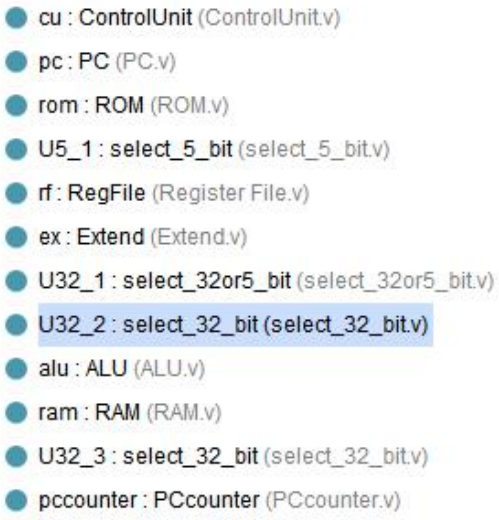


图 3 模块一览

3.4.1. ControlUnit：控制单元

最主要模块，控制各种信号以及根据指令代码选择 op 指令

3.4.2. PC&PCcounter:程序计数器

负责根据 PCSrc 进行下一指令地址的选择

3.4.3. ROM：指令存储器

addr，指令存储器地址输入端口

DataOut，指令存储器数据输出端口（指令代码输出端口）

rd，指令存储器读写控制信号，为 0 写，为 1 读

3.4.4. RAM：数据存储器

address，数据存储器地址输入端口

writeData，数据存储器数据输入端口

DataOut，数据存储器数据输出端口

nRD，数据存储器读控制信号，为 0 读

nWR，数据存储器写控制信号，为 0 写

3.4.5. Register File：寄存器组

ReadReg1，rs 寄存器地址输入端口

ReadReg2，rt 寄存器地址输入端口

WriteReg，将数据写入的寄存器端口，其地址来源 rt 或 rd 字段

WriteData，写入寄存器的数据输入端口

ReadData1，rs 寄存器数据输出端口

ReadData2，rt 寄存器数据输出端口

RegWre，写使能信号，为 1 时，寄存器可写

3.4.6. ALU:算术逻辑单元

ALUopcode,功能选择信号

rega,regb,输入运算数据

result,运算结果

Zero，运算结果为零，zero 为 1

3.4.7. Extend:立即数扩展

将立即数根据 ExtSel 进行 0 扩展或符号扩展

3.4.8. select_5_bit：选择模块 1

rd,rt 二选一，作为存储运算结果的寄存器地址

3.4.9. select_32or5_bit:选择模块 2

从 ReadData1 和 sa 中选择一个数据送入 ALU

3.4.10. select_32_bit:选择模块 3 和 4

从 ReadData2 和扩展后的立即数中，选择一个数据送入 ALU

从 ALU 结果 result 和 RAM 中数据二选一，结果存入寄存器 reg

3.5. 单周期 CPU 数据通路

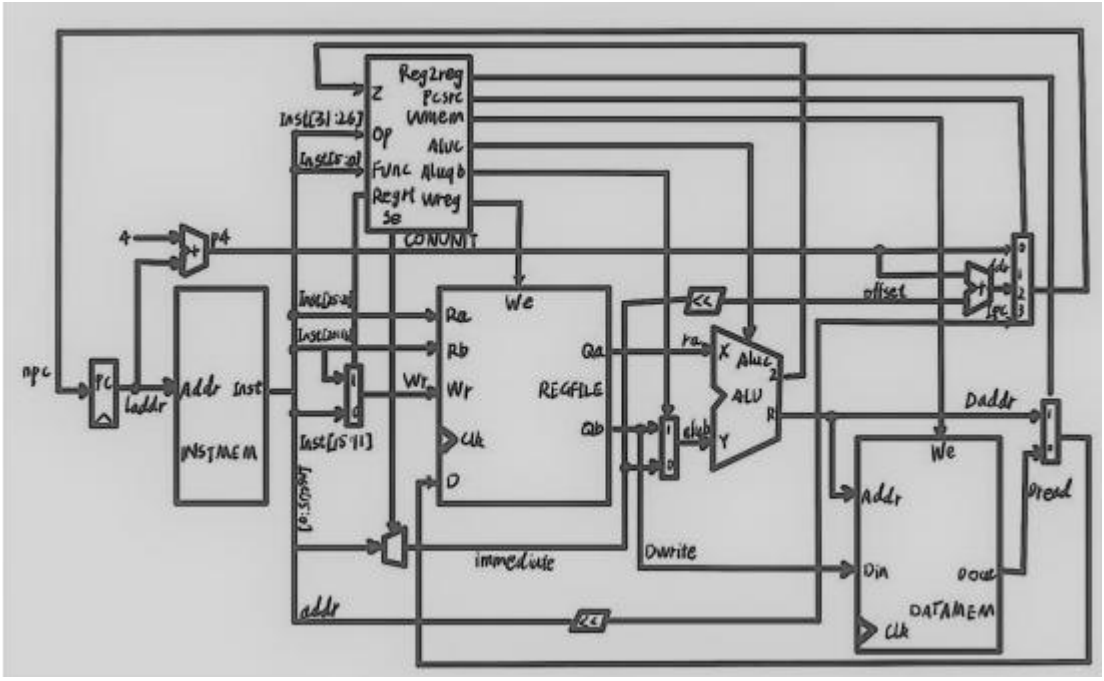


图 4CPU 模块连接图

3.6. 实现指令

指令功能	op				
add rd , rs, rt	000000	rs(5 位)	rt(5 位)	rd(5 位)	reserved
addi rt, rs , immediate	000001	rs(5 位)	rt(5 位)	immediate(16 位)	
sub rd , rs , rt	000010	rs(5 位)	rt(5 位)	rd(5 位)	reserved
ori rt , rs , immediate	010000	rs(5 位)	rt(5 位)	immediate(16 位)	
and rd , rs , rt	010001	rs(5 位)	rt(5 位)	rd(5 位)	reserved
or rd , rs , rt	010010	rs(5 位)	rt(5 位)	rd(5 位)	reserved
sll rd, rt,sa	011000	未用	rt(5 位)	rd(5 位)	sa reserved
slti rt, rs, immediate	011011	rs(5 位)	rt(5 位)	immediate(16 位)	
sw rt , immediate(rs)	100110	rs(5 位)	rt(5 位)	immediate(16 位)	
lw rt , immediate(rs) 读存储器	100111	rs(5 位)	rt(5 位)	immediate(16 位)	
beq rs,rt, immediate	110000	rs(5 位)	rt(5 位)	immediate(16 位)	
bne rs,rt, immediate	110001	rs(5 位)	rt(5 位)	immediate(16 位)	
j addr	111000	addr[27..2]			
halt	111111	00000000000000000000000000(26 位)			

表 3 实现指令

4. 具体实现代码(本人负责部分)

ControlUnit：控制单元

主要功能:根据指令代码选择 op 指令,控制各种信号，使得其他功能有效与无效

实现思路:根据输入的指令确定指令类型和指令名，并将控制信号的 0 、 1 通过逻辑运算与指令编码结合，来确定应当使用的部件与指令。

输入输出引脚及控制信号:

input [5:0] ：op 输入信号。

input zero：输入信号，判断 ALU 运算结果是否为 0，为 1 时，说明结果为 0。

Reset：重置信号，为 1 保持非重置。

PCWre：PC 改变信号，1 可变，0 不变。

ALUSrcA：选择信号，寄存器或者 sa 的值，二选一送入 ALU，作为运算值 A，为 0 选择前者，为 1 选择后者。

ALUSrcB：选择信号，寄存器或者扩展后立即数的值，二选一送入 ALU，作为运算值 B 为 0 选择前者，为 1 选择后者。

DBDataSrc：选择写回到寄存器的值，为 0 时，选择 ALU 的运算结果写入，为 1 时，选择从 RAM 读取数据写入。

RegWre：寄存器可写信号，为 1 可写。

InsMemRW：指令可读信号，为 1 可读。

RegDst：选择 rd 或者 rt 作为结果存储写入的位置，为 0 时，选择前者，为 1 时选择后者。

ExtSel：为 0 时，对立即数做 0 扩展，为 1 时，对立即数做符号扩展。

PCSrc[1:0]：为 00 时，顺序执行取指令，为 01 时，条件跳转执行，为 10 时，无条件跳转执行。

[2:0]ALUOp：决定 ALU 做何种运算。

```
module ControlUnit(
    input [5:0] op,
    input zero,
    output reg Reset,
    output reg PCWre,
    output reg ALUSrcA,
    output reg ALUSrcB,
    output reg DBDataSrc,
    output reg RegWre,
    output reg InsMemRW,
    output reg mRD,
    output reg mWR,
    output reg RegDst,
    output reg ExtSel,
    output reg [1:0] PCSrc,
    output reg [2:0] ALUOp
    );

    initial
        begin
            //Reset = 1;
            PCWre = 1;
            ALUSrcA = 0;
```

```
            ALUSrcB = 0;
            DBDataSrc = 0;
            RegWre = 0;
            InsMemRW = 1;
            mRD = 0;
            mWR = 0;
            RegDst = 0;
            ExtSel = 0;
            PCSrc = 0;
            ALUOp = 0;
        end
    always@(op or zero)
    begin
        case(op)
            //add
            6'b000000:
                begin
                    Reset = 1; //重置信号
                    PCWre = 1;//PC 改变（next+4）
                    ALUSrcA = 0;//选择寄存和 sa
                    ALUSrcB = 0;//寄存、扩展立即数
                    DBDataSrc = 0;//选择输出数据还是地址（0 是数据，1 地
址）
                    RegWre = 1;//寄存器可写
                    mWR = 0;//只与写数据寄存器（sw）有关（1）
                    RegDst = 1;//存 rt\rd
                    ALUOp = 000;//选择功能
                    PCSrc = 00;//决定下一个地址是否跳转
                end

            //addi
            6'b000001:
                begin
                    Reset = 1;
                    PCWre = 1;
                    ALUSrcA = 0;
                    ALUSrcB = 1;
                    DBDataSrc = 0;
                    RegWre = 1;
                    mWR = 0;
                    RegDst = 0;
                    ExtSel = 1;
                    ALUOp = 000;
                    PCSrc = 00;
```

```
        end

    //sub
    6'b000010:
        begin
            Reset = 1;
            PCWre = 1;
            ALUSrcA = 0;
            ALUSrcB = 0;
            DBDataSrc = 0;
            RegWre = 1;
            mWR = 0;
            RegDst = 1;
            ALUOp = 001;
            PCSrc = 00;
        end

    //ori
    6'b010000:
        begin
            Reset = 1;
            PCWre = 1;
            ALUSrcA = 0;
            ALUSrcB = 1;
            DBDataSrc = 0;
            RegWre = 1;
            mWR = 0;
            RegDst = 0;
            ExtSel = 0;
            ALUOp = 011;
            PCSrc = 00;
        end

    //and
    6'b010001:
        begin
            Reset = 1;
            PCWre = 1;
            ALUSrcA = 0;
            ALUSrcB = 0;
            DBDataSrc = 0;
            RegWre = 1;
            mWR = 0;
            RegDst = 1;
```

```
            ALUOp = 100;
            PCSrc = 00;
        end

    //or
    6'b010010:
        begin
            Reset = 1;
            PCWre = 1;
            ALUSrcA = 0;
            ALUSrcB = 0;
            DBDataSrc = 0;
            RegWre = 1;
            mWR = 0;
            RegDst = 1;
            ALUOp = 011;
            PCSrc = 00;
        end

    //sll
    6'b011000:
        begin
            Reset = 1;
            PCWre = 1;
            ALUSrcA = 1;
            ALUSrcB = 0;
            DBDataSrc = 0;
            RegWre = 1;
            mWR = 0;
            RegDst = 1;
            ALUOp = 010;
            PCSrc = 00;
        end

    //slti
    6'b011011:
        begin
            Reset = 1;
            PCWre = 1;
            ALUSrcA = 0;
            ALUSrcB = 1;
            DBDataSrc = 0;
            RegWre = 1;
            mWR = 0;
```

```
            RegDst = 0;
            ExtSel = 1;
            ALUOp = 101;
            PCSrc = 00;
        end

//sw
6'b100110:
        begin
            Reset = 1;
            PCWre = 1;
            ALUSrcA = 0;
            ALUSrcB = 1;
            RegWre = 0;
            mWR = 1;
            ExtSel = 1;
            PCSrc = 00;
            ALUOp = 000;
        end

//lw
6'b100111:
        begin
            Reset = 1;
            PCWre = 1;
            ALUSrcA = 0;
            ALUSrcB = 1;
            DBDataSrc = 1;
            RegWre = 0;
            mRD = 1;
            mWR = 0;
            RegDst = 0;
            ExtSel = 1;
            PCSrc = 00;
            ALUOp = 000;
        end

//beg
6'b110000:
        begin
            Reset = 1;
            PCWre = 1;
            ALUSrcA = 0;
            ALUSrcB = 0;
```

```
            DBDataSrc = 0;
            RegWre = 0;
            mWR = 0;
            ExtSel = 1;
            ALUOp = 001;
            if(zero == 1) begin
                        PCSrc = 1'b01;
                  end
            else begin
                        PCSrc = 1'b00;
                  end
      end

//bne
6'b110001:
      begin
            if(zero == 1) begin
                  PCSrc = 1'b00;
            end
      else begin
                  PCSrc = 1'b01;
            end
            Reset = 1;
            PCWre = 1;
            ALUSrcA = 0;
            ALUSrcB = 0;
            DBDataSrc = 0;
            RegWre = 0;
            mWR = 0;
            ExtSel = 1;
            ALUOp = 001;

      end

//j
6'b111000:
      begin
            Reset = 1;
            PCWre = 1;
            RegWre = 0;
            PCSrc = 10;
      end

//halt
```

```
                6'b111111:
                    begin
                        Reset = 0;
                        PCWre = 0;
                        ALUSrcB = 0;
                        RegWre = 0;
                    end
            endcase
        end
endmodule
```

4.1. select_5_bit：选择模块 1

主要功能：rd,rt 二选一，作为存储运算结果的寄存器地址。

实现思路：选择信号 control 控制选择，=0 时，选择 rd，=1 时，选择 rt。

```
module select_5_bit(
    input [4:0] select_one,//rd
    input [4:0] select_two,//rt
    input control,//1 是 rt，0 是 rd
    output [4:0] result
    );
    assign result = (control == 1'b0 ? select_one : select_two);
endmodule
```

4.2. select_32or5_bit:选择模块 2

主要功能：从 ReadData1 和 sa 中选择一个数据送入 ALU

实现思路：选择信号 control 控制选择，=0 时，选择 ReadData1，=1 时，选择 sa。

```
module select_32or5_bit(
    input [31:0] select_one,
    input [4:0] select_two,
    input control,
    output [31:0] result
    );
    wire[31:0] select_three;
    assign select_three[4:0] = select_two;
    assign select_three[31:5] = 1'b00000000000000000000000000000;
    assign result = (control == 1'b0 ? select_one : select_three);
endmodule
```

4.3. select_32_bit:选择模块 3

主要功能：从 ReadData2 和扩展后的立即数中，选择一个数据送入 ALU。从 ALU 结果 result 和 RAM 中数据二选一，结果存入寄存器 reg

实现思路：选择信号 control 控制选择，=0 时，选择 ReadData1，=1 时，选择 sa。用于后者时，=0 时，选择 result，=1 时，选择 RAM。

```
module select_32_bit(
    input [31:0] select_one,
    input [31:0] select_two,
    input control,
    output [31:0] result
    );
    assign result = (control == 1'b0 ? select_one : select_two);
endmodule
```

4.4. ALU

主要功能：接受输入的数据并根据信号进行相应计算。
实现思路：定义八种运算方式，每一种对应一个 aluop 码，根据输入的 aluop 的值决定做何种运算。

```
module ALU(
    input [2:0] ALUopcode,
    input [31:0] rega,
    input [31:0] regb,
    output reg [31:0] result,
    output zero
    );
    initial
    begin
        result = 0;
    end
    assign zero = (result==0)?1:0;
    always @( ALUopcode or rega or regb ) begin
        case (ALUopcode)
            3'b000 : result = rega + regb;
            3'b001 : result = rega - regb;
            3'b010 : result = regb << rega;
            3'b011 : result = rega | regb;
            3'b100 : result = rega & regb;
            3'b101 : result = (rega < regb)?1:0; // 不带符号比较
            3'b110 : begin // 带符号比较
                if(rega < regb && (rega[31] == regb[31]))result = 1;
```

```
                else if (rega[31] == 1 && regb[31] == 0) result = 1;
                else result = 0;
                end
            3'b111 : result = rega ^ regb;
            default : begin
                result = 32'h00000000;
                $display (" no match");
                end
        endcase
    end
endmodule
```

4.5. Extend

主要功能：实现立即数扩展

实现思路：根据选择信号，为 0 时，进行 0 扩展，为 1 时进行立即数扩展

```
module Extend(
    input [15:0] immediate,
    input ExtSel,
    output reg[31:0] outData
    );

    always@(immediate or ExtSel)
    begin
        case(ExtSel)
            1'b0:
                begin
                    outData[15:0] = immediate;
                    outData[31:16] = 16'h0000;
                end
            1'b1:
                begin
                    outData[15:0] = immediate;
                    outData[31:16] = (immediate[15])? 16'hffff : 16'h0000;
                end
        endcase
    end
```

4.6. PCcounter

主要功能：选择下一条指令执行顺序。

实现思路：输入选择信号，为 00 时顺序执行，为 01 时条件跳转执行，为 10 时，无条件跳转执行。

```
module PCcounter(
```

```
    input [1:0] PCSrc,
    input [31:0] currentAddress,
    output reg [31:0] newAddress,
    input [31:0] outData,
    input [25:0] jAddress
    );

    wire [31:0] temp_one, temp_two, temp_three;
    assign temp_one = currentAddress + 4;
    assign temp_two[25:0] = jAddress[25:0];
    assign temp_three = temp_two << 2;
    always@(PCSrc or currentAddress or outData or jAddress or temp_one or
temp_three )
        begin
            case(PCSrc)
                2'b00:    newAddress = currentAddress+4;
                2'b01:
                    begin
                        newAddress = (currentAddress+4)+ (outData << 2);
                        newAddress[1:0] = 1'b00;
                    end

                2'b10:
                begin
                    newAddress[31:28] = temp_one[31:28];
                    newAddress[27:0] = temp_three[27:0];
                end
             endcase
        end
endmodule
```

5. 仿真结果

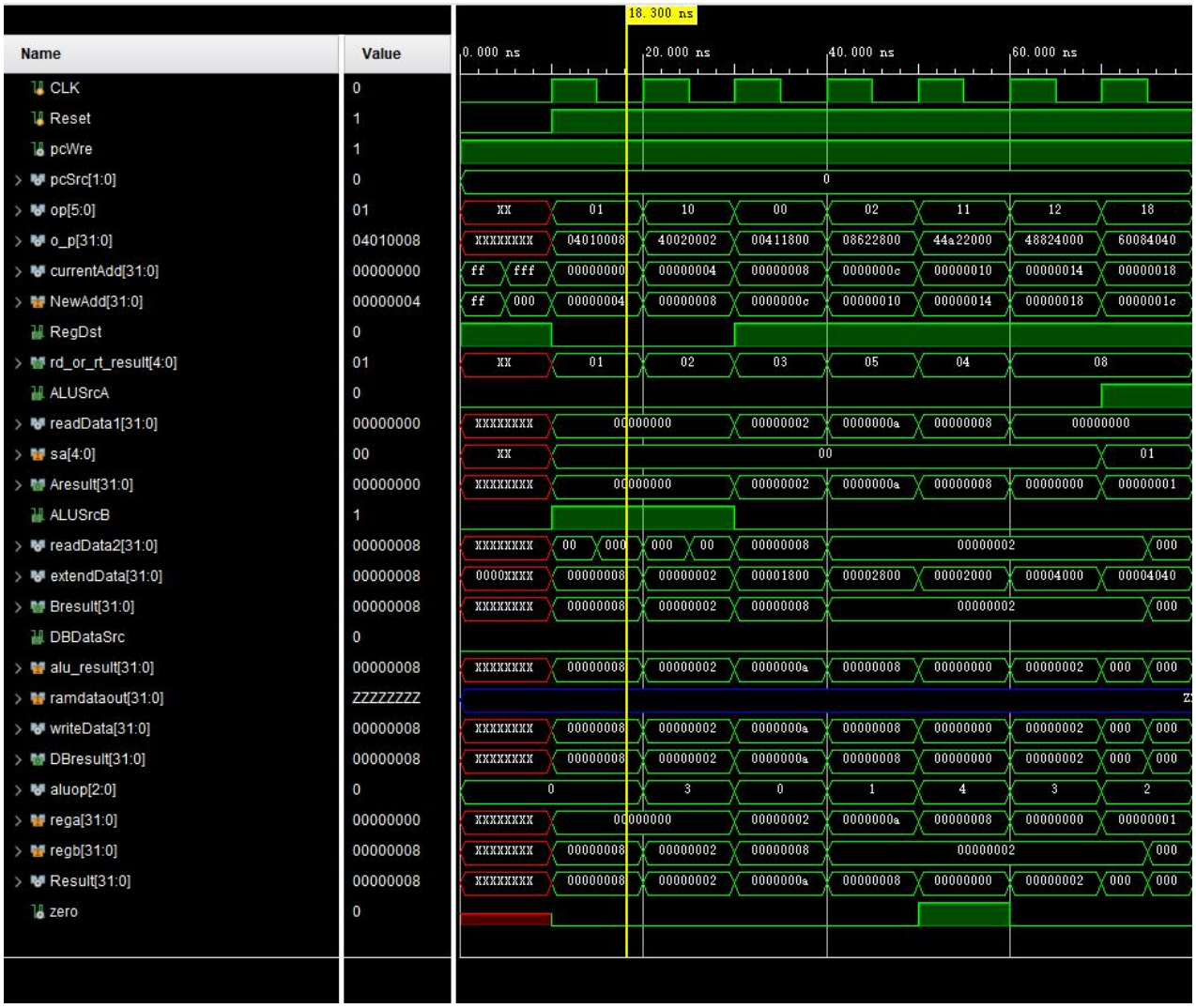


图 5 仿真模拟图前半部分

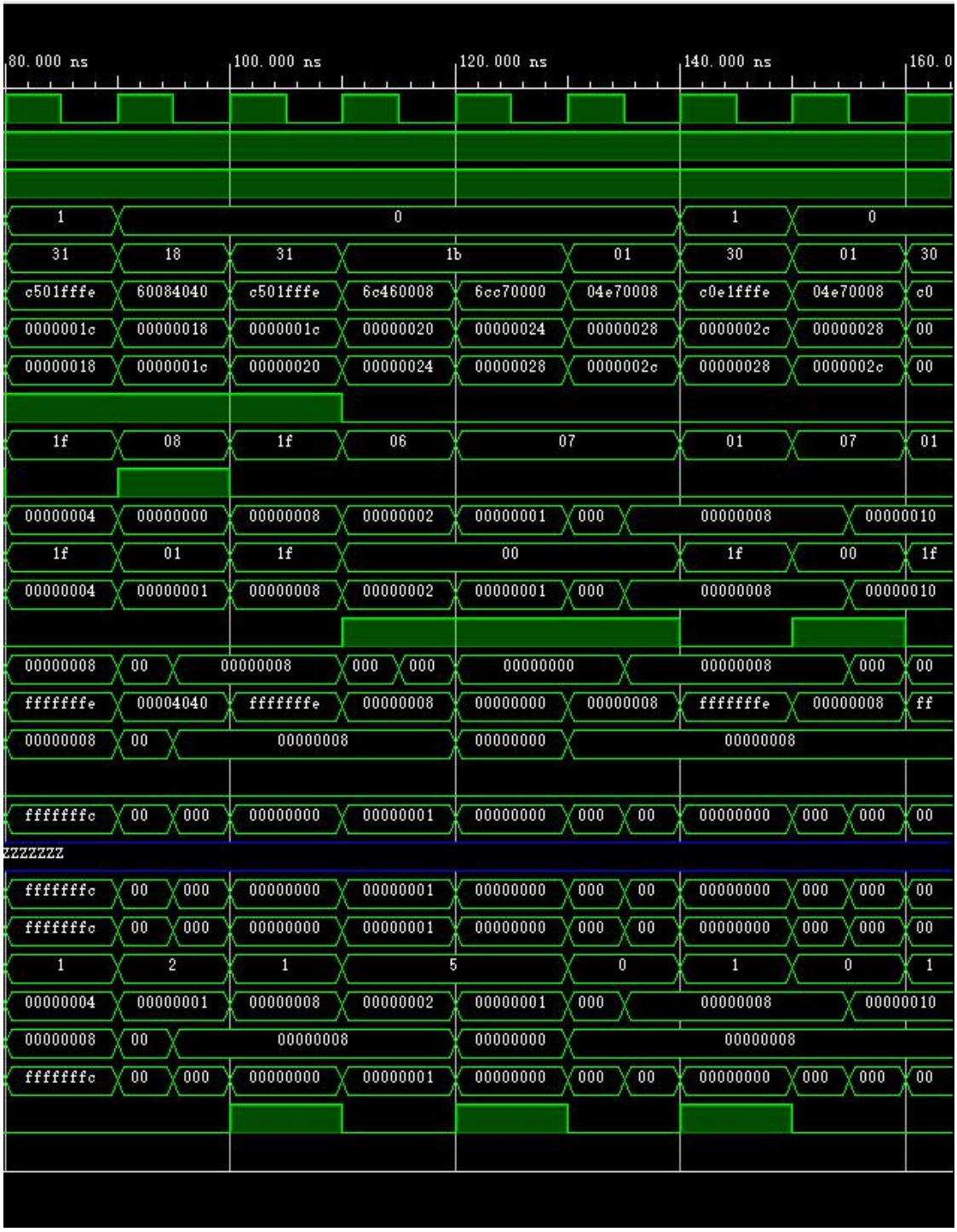
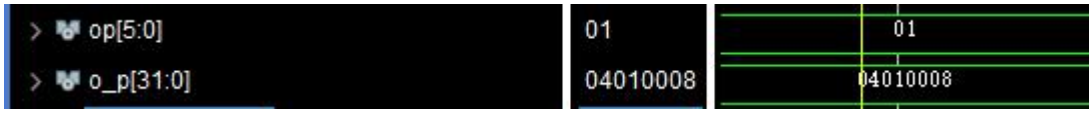


图 6 仿真模拟图后半部分

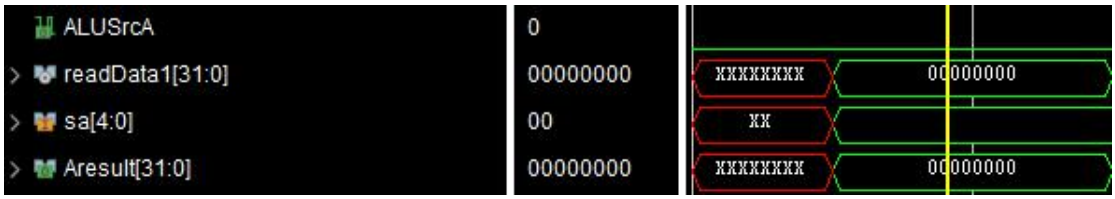
5.1.1. 第一条指令 ADDI

由 ROM 在本地文件读入，读入后，根据位数拆分成为 op，rs，rt，rd，立即数等。op 送入 CU 模块，进一步选择具体操作。

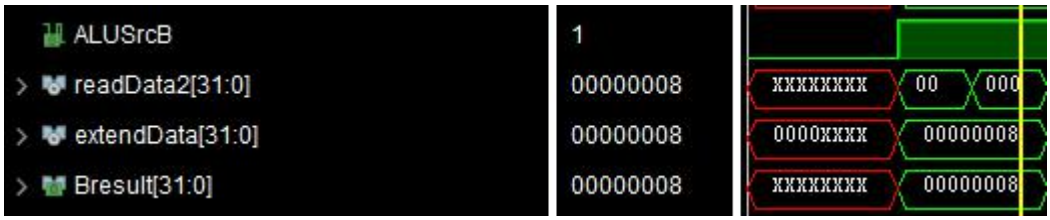


op 为 000001，做 addi 运算，该指令实现立即数加法。

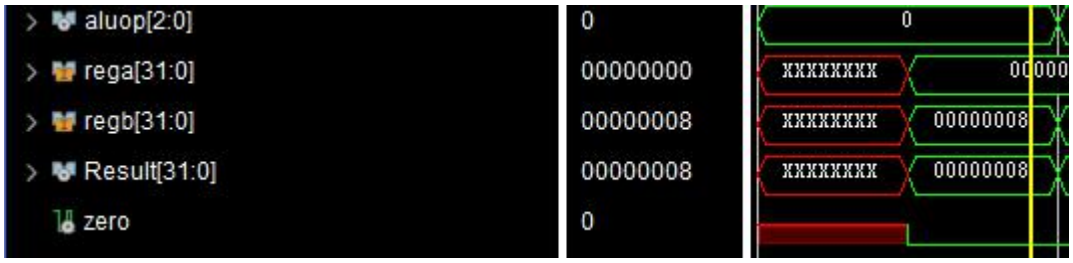
rd 和 rt 送入五位选择模块，选择出最后运算结果是要存入 rt 地址寄存器中。



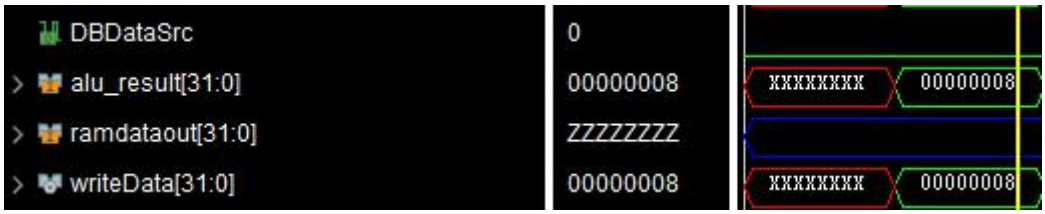
根据选择信号 ALUSrcA，选择从寄存器中取数据作为 ALU 的一个运算元。



根据选择信号 ALUSrcB，选择扩展后的立即数，作为 ALU 的一个运算元。



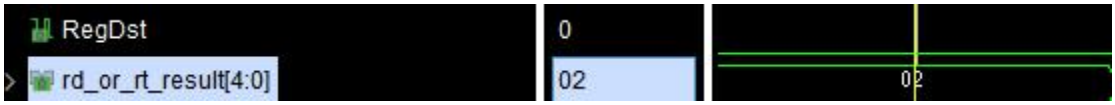
aluop 由 CU 模块中 op 指令决定，进行哪种运算，rega+regb=result，结果不为零，zero=0。



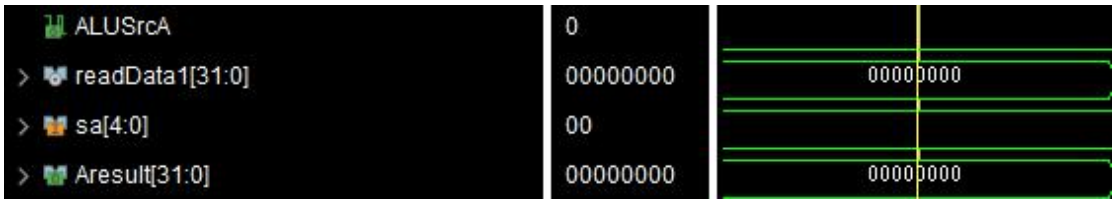
根据 DBDataSrc，选择 alu 运算后的结果作为 writeData 存回到 rd 地址对应的寄存器中。

5.1.2. 第二条指令 ORI

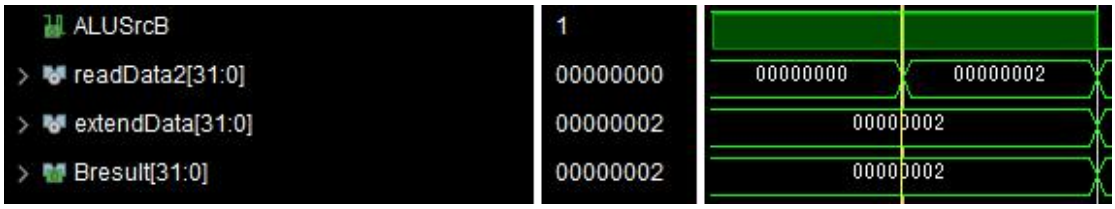
Op=000010 该指令实现立即数或运算。



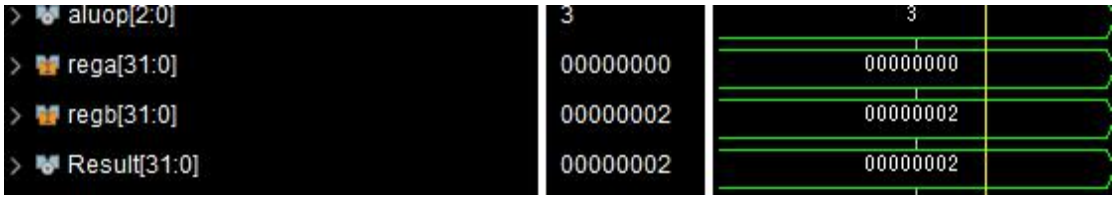
结果存入 rt 寄存器。



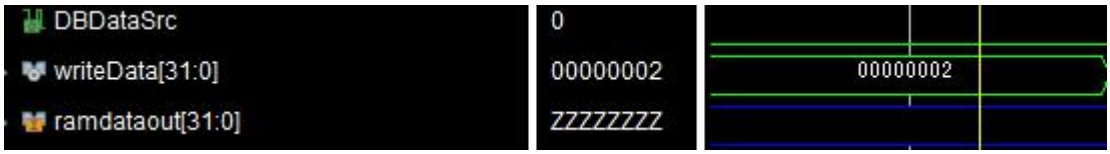
从寄存器取值，Aresult 为选择的结果，等于 readData1，送入 ALU。



从寄存器取值，Bresult 为选择的结果，等于 readData2，送入 ALU。



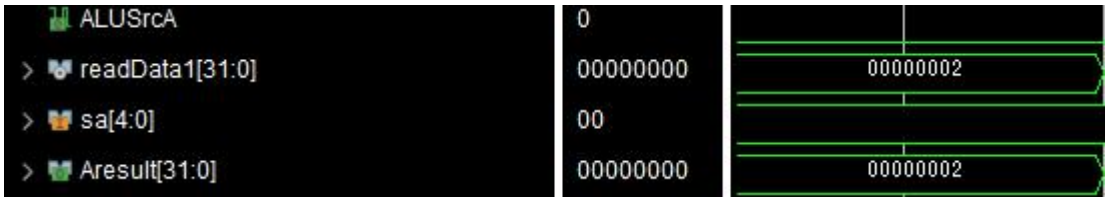
Aluop=011，执行或运算，rega|regb=00000002



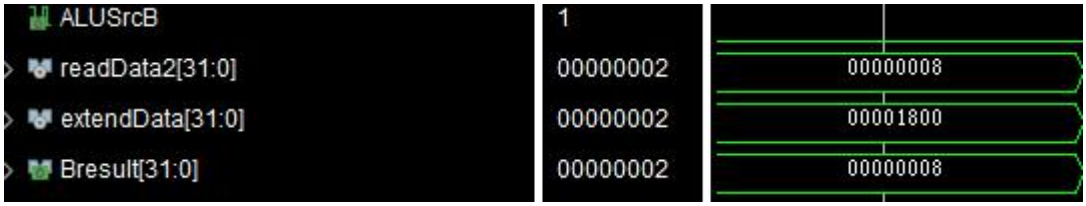
结果写回到寄存器。

5.1.3. 第三条指令 ADD

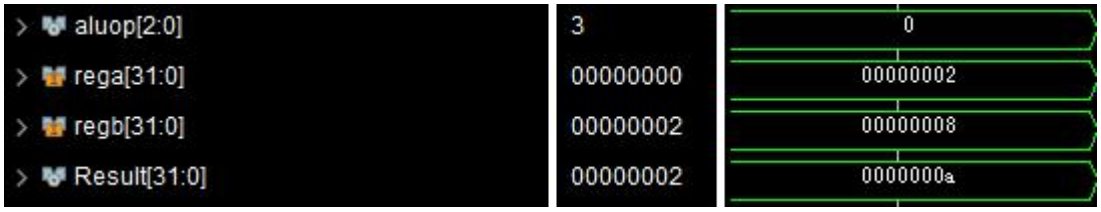
Op=000000 执行加法运算。



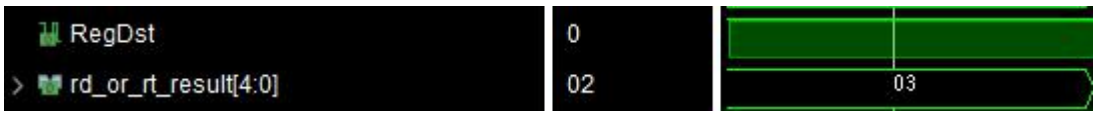
选择从寄存器中取值，readData1 送入 ALU 中运算。



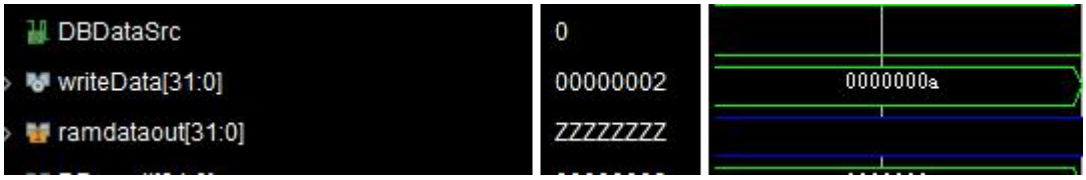
选择从寄存器中取值，readData2 送入 ALU 中运算。



Aluop=0，做加运算，rega+regb=2+8=10



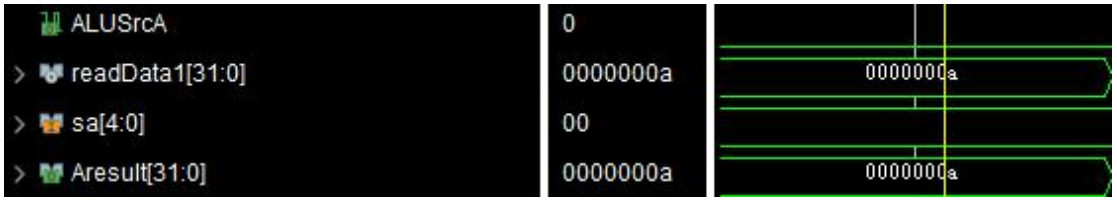
Regdst=1，选择 rd 作为结果保存所在地址



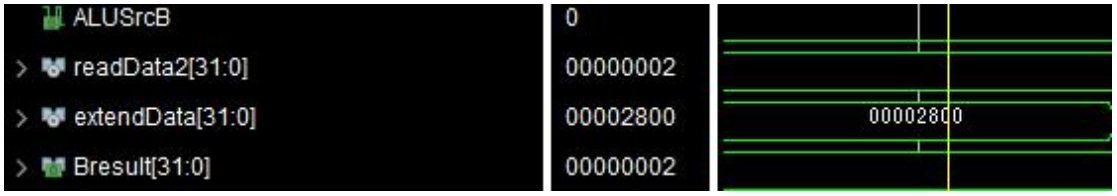
结果写回到 rd 寄存器中。

5.1.4. 第四条指令 SUB

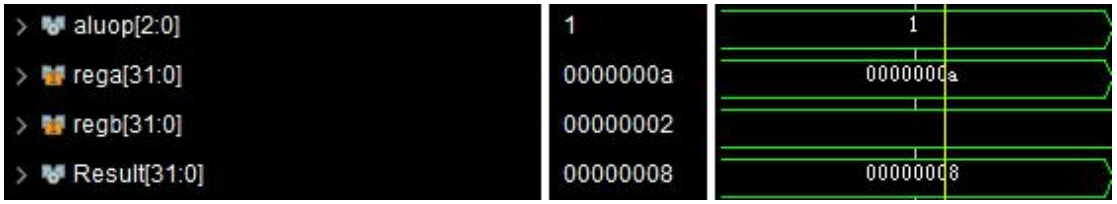
Op=000010，执行减法运算



选择从寄存器中取值，readData1 送入 ALU 中运算。

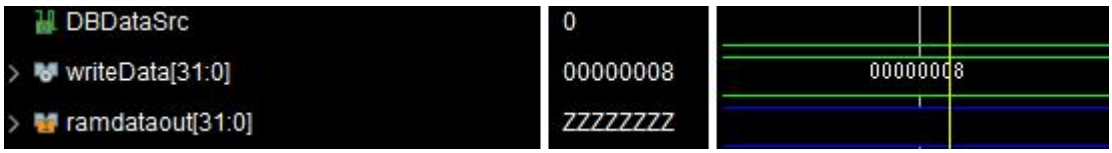


选择从寄存器中取值，readData2 送入 ALU 中运算。



Aluop=001，执行减法运算，rega-regb=10-2=8

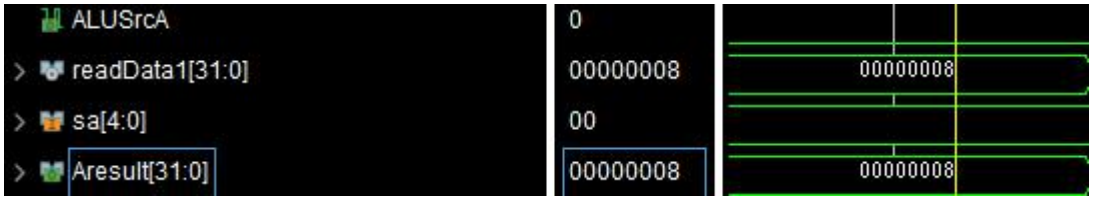
Regdst=1，选择 rd 作为结果回写到寄存器内的地址。



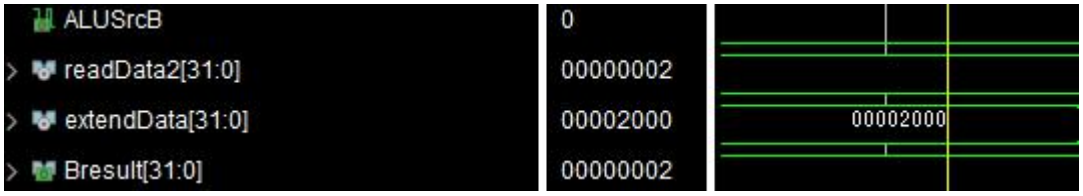
选择 writeData 作为回写数据。

5.1.5. 第五条指令 AND

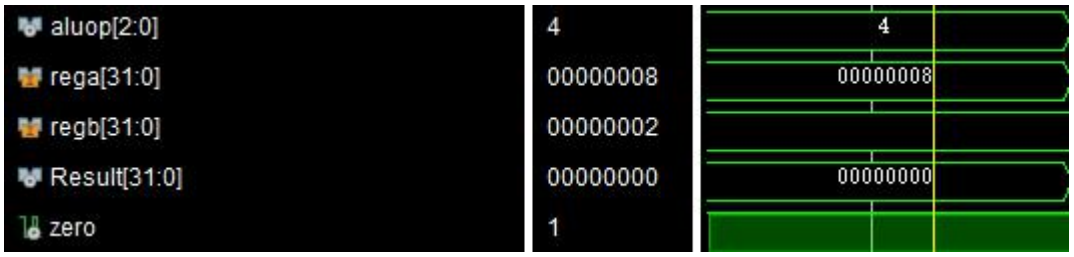
Op=010001 执行与运算



选择从寄存器中取值，readData1 送入 ALU 中运算。

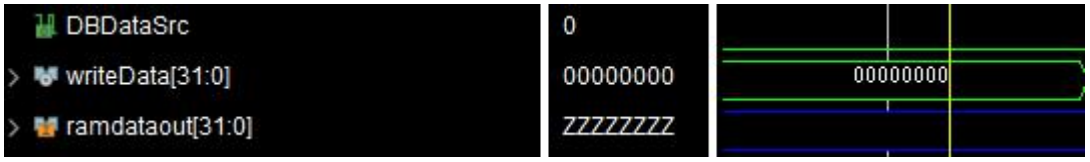


选择从寄存器中取值，readData2 送入 ALU 中运算。



Aluop=100，rega&regb=0，result=0，所以 zero=1。

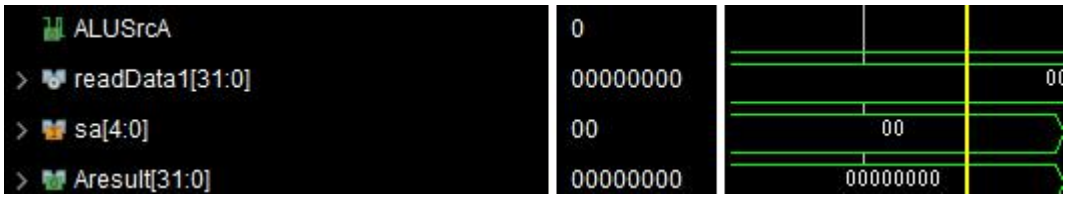
Regdst=1，选择 rd 作为结果回写到寄存器内的地址。



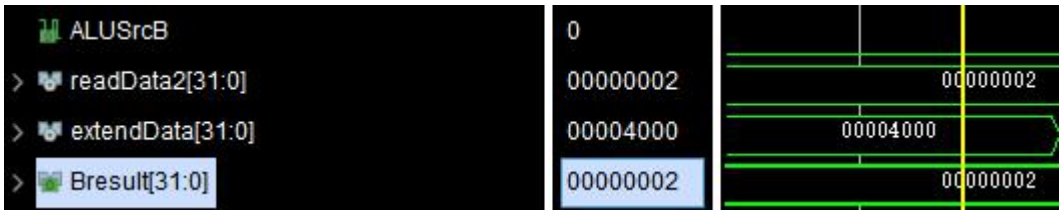
选择将 ALU 结果写回到寄存器。

5.1.6. 第六条指令 OR

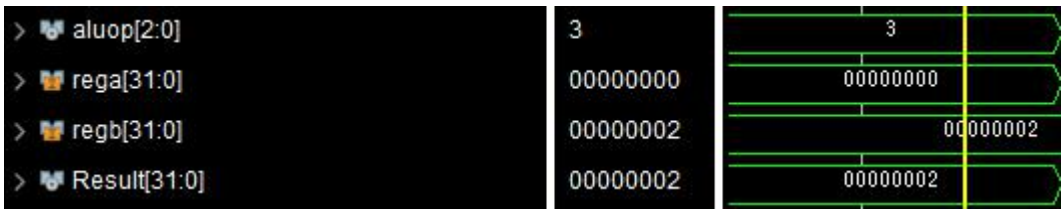
Op=010010，执行或运算



选择从寄存器中取值，readData1 送入 ALU 中运算。

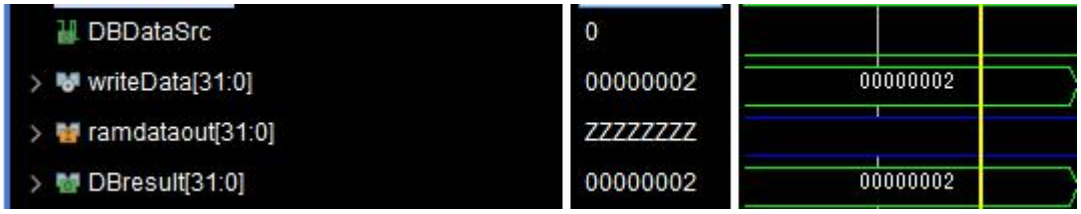


选择从寄存器中取值，readData2 送入 ALU 中运算。



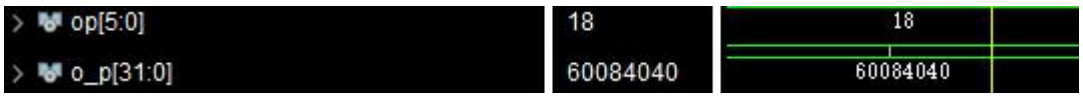
Aluop=011，执行或运算，rega|regb=2

Regdst=1，选择 rd 作为结果回写到寄存器内的地址。

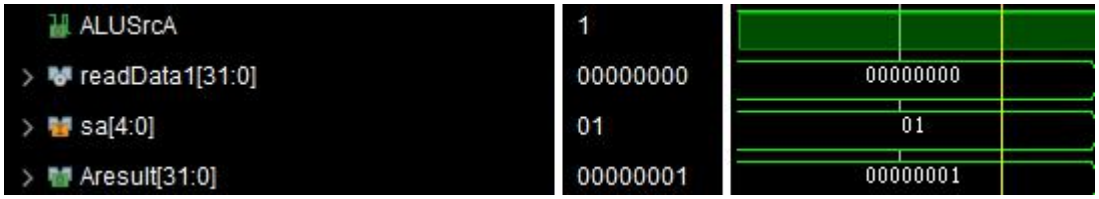


DBDataSrc=0，选择 writeData 作为写回到寄存器的值。

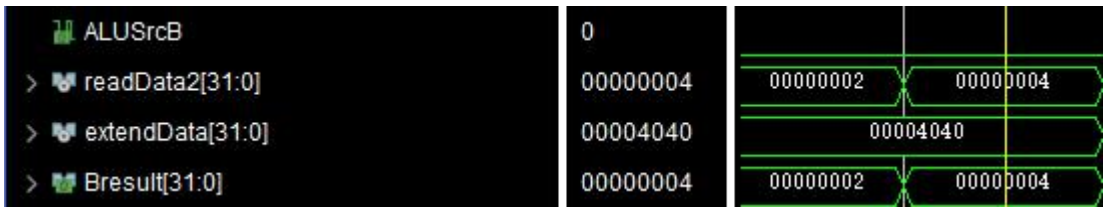
5.1.7. 第七条指令 SLL



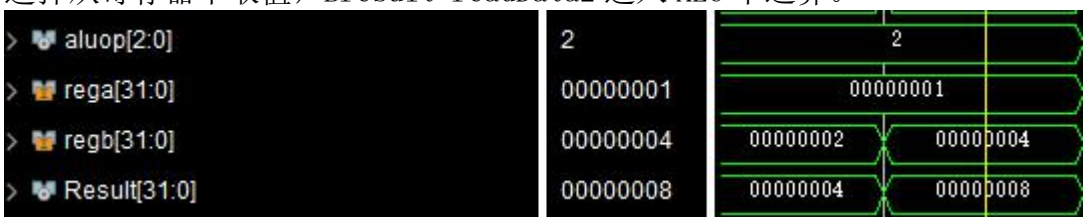
Op=011000，执行位移运算。



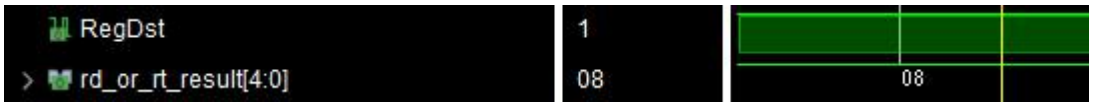
ALUSrcA=1，选择 sa 的值作为 Aresult 送入 ALU 运算。



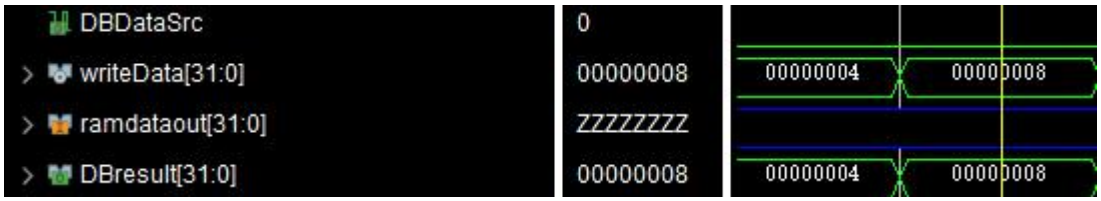
选择从寄存器中取值，Bresult=readData2 送入 ALU 中运算。



Aluop=010，执行移位运算，4 左移一位运算的结果为 result=8。

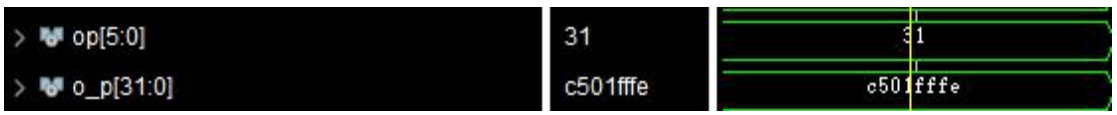


Regdst=1，选择 rd 作为结果回写到寄存器内的地址。



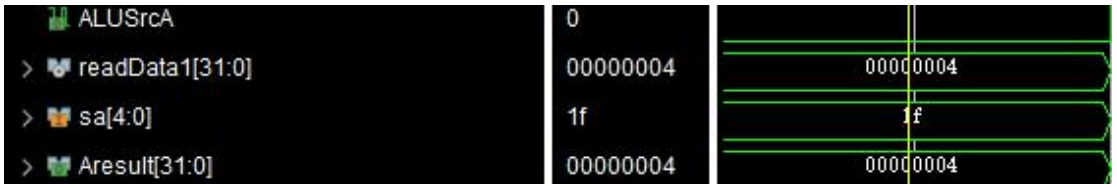
DBDataSrc=0，选择 writeData 作为写回到寄存器的值。

5.1.8. 第八条指令 BNE



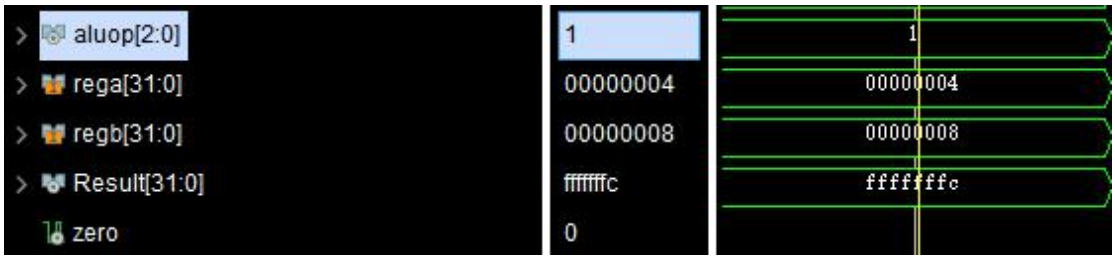
Op=110001，条件跳转指令

Pcsrc=01 说明执行条件跳转



ALUSrcA=0,选择从寄存器中取值，Aresult=readData1 送入 ALU 中运算。

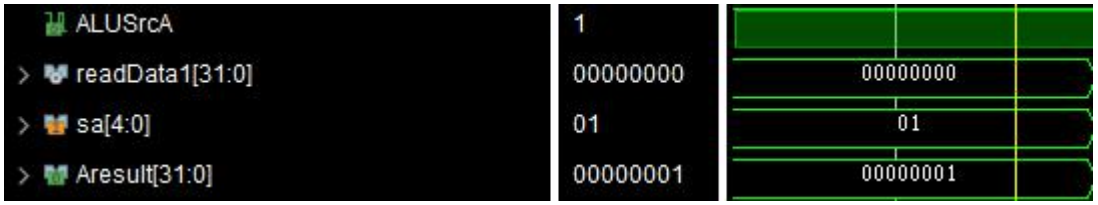
ALUSrcB=0,选择从寄存器中取值，Bresult=readData2 送入 ALU 中运算。



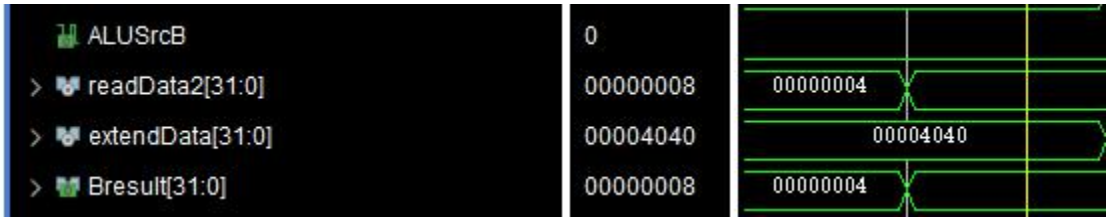
Aluop=1，rega-regb 不等于 0，即 rega 和 regb 不等，发生跳转，跳转到上一条执行过的指令。

5.1.9. 第九条指令 SLL

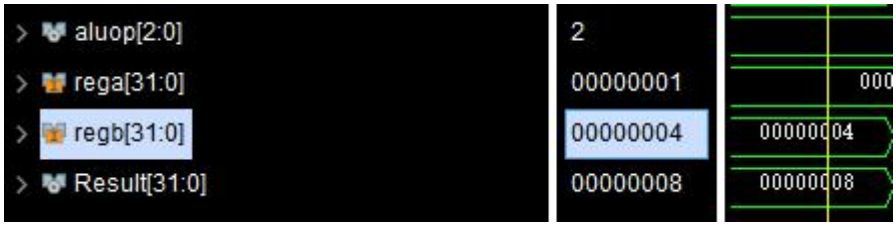
Op=011000，执行位移运算。



ALUSrcA=1，选择 sa 的值作为 Aresult 送入 ALU 运算。



ALUSrcB=0，选择从寄存器中取值，Bresult=readData2 送入 ALU 中运算。



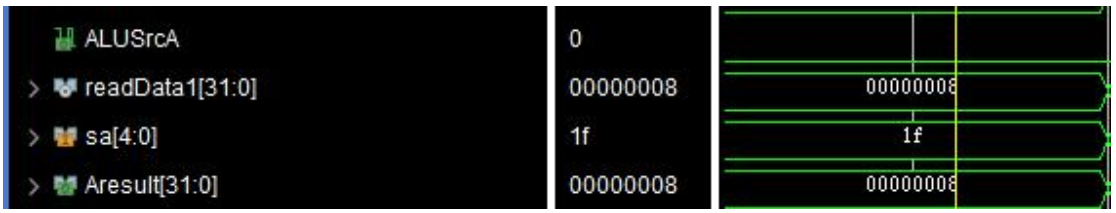
Aluop=010，Regb<<rega，结果为 4 左移一位，result=8。

Regdst=1，选择 rd 作为结果回写到寄存器内的地址。

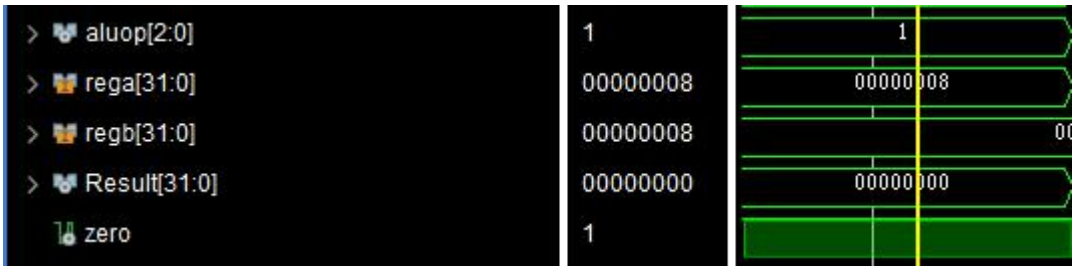
DBDataSrc=0，选择 writeData 作为写回到寄存器的值。

5.1.10. 第十条指令 BNE

再次执行到 bne 条件跳转指令，由于和第七条指令相同，所以只在此分析结果。

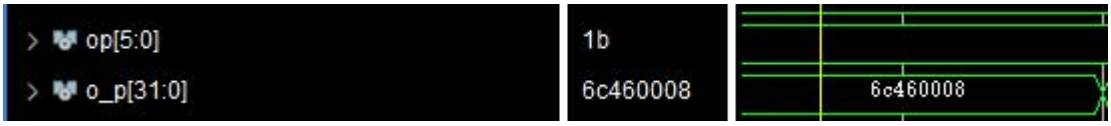


不同点在于，这次从寄存器中读取的数值，为上一步进行位移运算后存储到寄存器的值 ReadData=8。

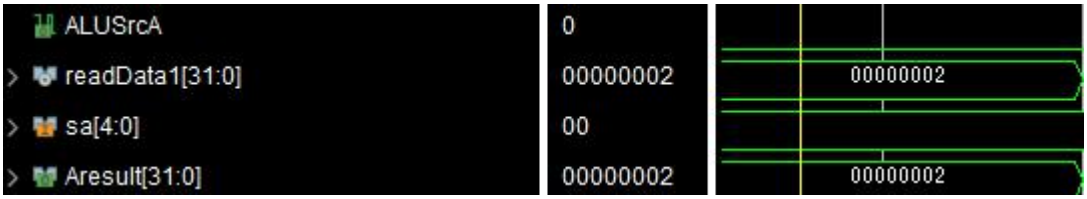


Aluop=001，rega-regb=0，zero=0，所以相等，不再进行条件跳转，而是顺序执行指令。

5.1.11. 第十一条指令 SLTI

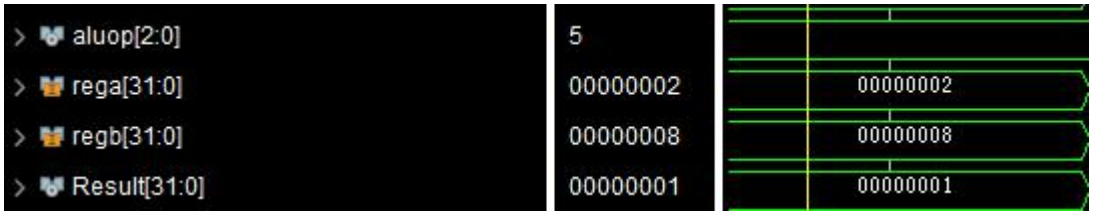


Op=011011，执行立即数比较指令。

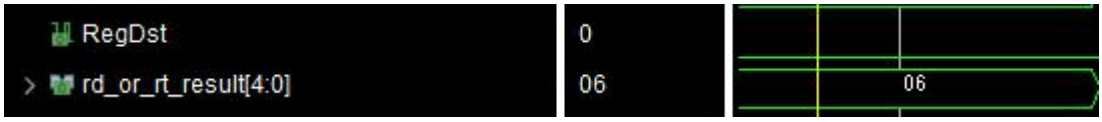


ALUSrcA=0，选择从寄存器中取值，Aresult=readData2 送入 ALU 中运算。

ALUSrcB=1，选择扩展立即数，Bresult=extend 送入 ALU 中运算。



Aluop=101，执行无符号比较，rega<regb，所以 result=1。

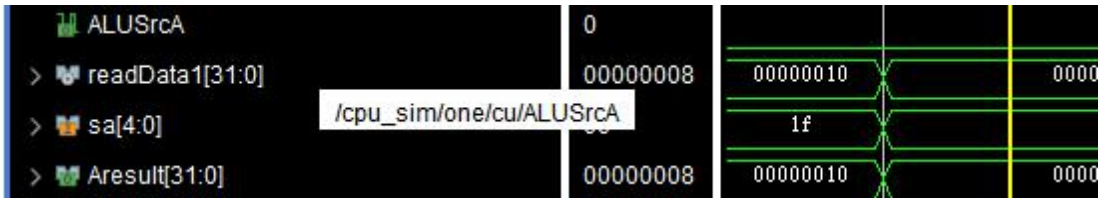


Regdst=0，选择 rt 作为结果回写到寄存器内的地址,即 rt=1。

DBDataSrc=0，选择 writeData 作为写回到寄存器的值。

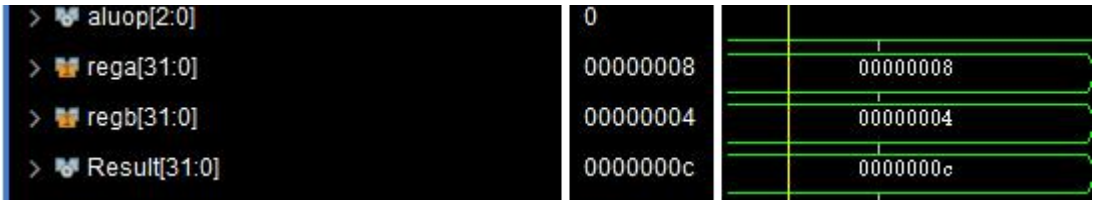
5.1.12. 第十二条指令 SW

Op=100110，执行 sw 指令，该指令执行将 rt 寄存器的内容保存到 rs 寄存器内容和立即数符号扩展后的数相加作为地址的内存单元中。



Rs 寄存器内容为 readData1=8

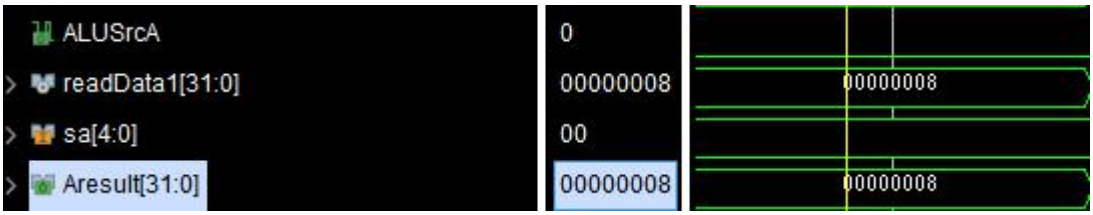
扩展后的立即数为 4



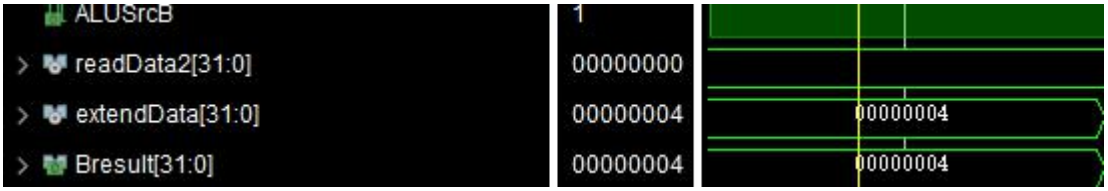
相加后为 12 作为地址，将 rt 中的值 02 存储到 RAM 中，从第 12 号寄存器开始

5.1.13. 第十三条指令 LW

即读取 rs 寄存器内容和立即数符号扩展后的数相加作为地址的内存单元中的数，然后保存到 rt 寄存器中



Rs 寄存器内容为 readData1=8

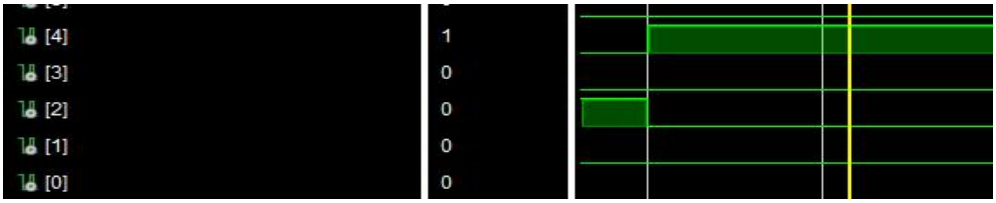


扩展后的立即数为 4，相加等于 12。将 RAM 中 12 号寄存器的值保存到 rt 寄存器中

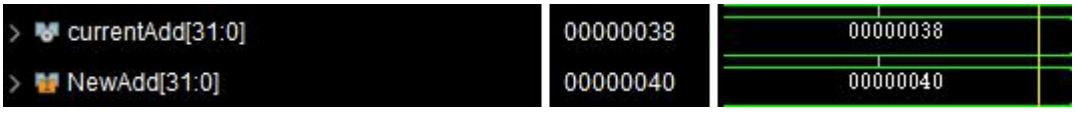
5.1.14. 第十四条指令 J

Op=111000，执行无条件跳转指令。

Pcsrc=10，执行无条件跳转



待跳转地址为指令后 26 位，为 0x00000040。

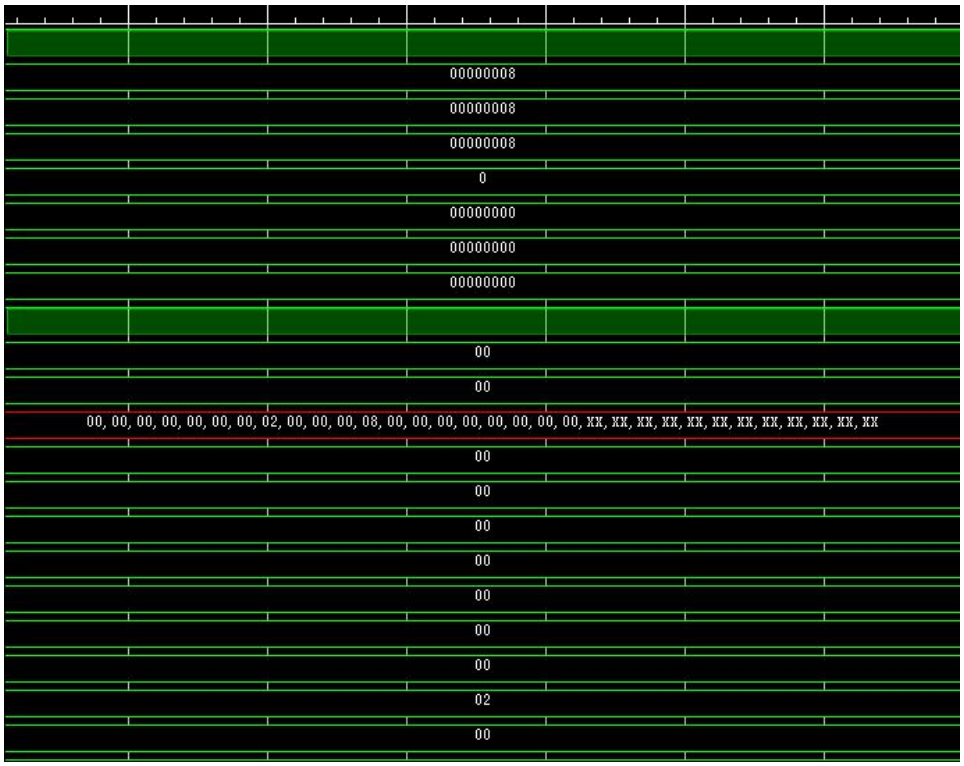


当前地址为 00000038，下一指令地址如果顺序执行为 00000042，现在为 00000040，说明跳转成立

5.1.15. 第十五条指令 HALT

Op=111111，halt 停机指令。

pcWre 置为 0，表示 PC 不再改变



6. 心得体会

本次课程设计做的是单周期 CPU，在上学期的实验中已经实现过部分功能，但本次的功能更加具体，需要对之前项目进行改进优化。首先通过对单周期 CPU 数据通路和控制线路图的学习了解，具体分析了各种指令的处理过程后，将 CPU 内各个部分模块化，分别实现一定的功能，然后通过对应的控制信号连接起来，最后用顶层模块将其联系成为一个整体。模块的敏感信号的选择都是不同的，有些模块要在时钟信号上升沿触发，而有些模块在下降沿触发， wire 与 reg 这两种变量也需要格外注意，wire 主要起连接各部分模块，化零为整的工具，由于不保存状态，它的值的随时可以改变，不受时钟信号的影响；而 reg 则是寄存器的抽象表达，可以用于存储数值，用于保留数据。相应的，我知道了 assign 是连续赋值，always 是敏感赋值。连续赋值，就是无条件全等。敏感赋值，就是有条件相等。assign 的对象是 wire，always 的对象是 reg。最终本次课程设计在基于上学期的基础上，进行了部分功能添加，最终达到了实验目的，在对指令进行验证的过程中，让我深入了解了数据在 CPU 中的流动，收获颇丰。